```
def matran(data):
    distances = euclidean_distances(data)
    return distances
X=df.head(5).values
print(X)
matran(X)
```

```
[[15 39]
 [15 81]
 [16  6]
 [16 77]
 [17 40]]

array([[ 0.        , 42.        , 33.01514804, 38.01315562,  2.23606798],
       [42.        ,  0.        , 75.00666637,  4.12310563, 41.0487515 ],
       [33.01514804, 75.00666637,  0.        , 71.        , 34.0147027 ],
       [38.01315562,  4.12310563, 71.        ,  0.        , 37.01351105],
       [ 2.23606798, 41.0487515 , 34.0147027 , 37.01351105,  0.        ]])
```

Chuyển đổi từ mảng thành matrix khoảng cách euclidean

```
def find(matrix):
    np.fill_diagonal(matrix, np.inf)
    min_distance = np.min(matrix)
    merged_point = np.unravel_index(np.argmin(matrix), matrix.shape)
    return min_distance, merged_point
find(matran(X))
```

```
(2.23606797749979, (0, 4))
```

Tìm kiếm khoảng cách nhỏ nhất của cặp điểm trong ma trận khoảng cách euclidean. Đầu ra trả về giá trị khoảng cách và vị trí của cặp điểm đó

```
def find_position(data, n):
    indices = np.argwhere(data == n)
    if len(indices) > 0:
        index = indices[0]
        return index
    else:
        return None
X1=X.copy()
print(X1)
find_position(X1, 77)
```

```
[[15 39]
 [15 81]
 [16  6]
 [16 77]
 [17 40]]

array([3, 1], dtype=int64)
```

Tìm kiếm vị trí của 1 số trong mảng

```
def upda(data, labels, cluster, clus):
    df = data.copy()
    matrix = matran(data)
    min_distance, merged_point = find(matrix)

    a = merged_point[0]
    b = merged_point[1]
    p = np.maximum(data[a], data[b])
    v,w = find_position(df, p)
    new_data = np.delete(df, v, axis=0)

    updated_points = (merged_point[0], merged_point[1])
```

Hàm cập nhật lại mảng
Đầu tiên chuyển mảng thành 1 ma trận khoảng cách
Tìm cặp điểm có giá trị thấp nhất , trả về vị trí cặp điểm trong ma trận và giá trị khoảng cách
Gán các điểm a,b là vị trí cặp điểm vừa tìm được
P là giá trị trả về giá trị lớn nhất

Tìm vị trí của giá trị P tìm được trong mảng đầu vào

Cập nhật lại mảng bằng cách xóa vị trí có giá trị lớn nhất của 2 cặp điểm là P . Đồng nghĩa việc giữ lại điểm có giá trị bé hơn

```
  # Gán cluster cho nhãn
  if cluster.get(labels[a]) is None and cluster.get(labels[b]) is None:
      cluster[labels[a]] = labels[a]
      cluster[labels[b]] = labels[a]
  elif cluster.get(labels[a]) is None:
      cluster[labels[a]] = cluster[labels[b]]
  elif cluster.get(labels[b]) is None:
      cluster[labels[b]] = cluster[labels[a]]
  else:
      cluster_b = cluster[labels[b]]
      for key, value in cluster.items():
          if value == cluster_b:
              cluster[key] = cluster[labels[a]]

  clus.append(list(cluster.values()).copy())
```

Dán nhãn từng điểm dữ liệu là biến labels
cluster chứa tất cả các dán nhãn cần được phân cụm
Nếu vị trí a,b không được chứa trong cluster thì gán 2 giá trị đó cùng 1 dán nhãn. Nếu a hoặc b chứa trong cluster thì gán giá trị đó bằng với giá trị cluster. Thực hiện cho đến khi mảng là rỗng

```
data = df.head(200).values
labels = np.arange(1, len(data) + 1)
cluster = {label: label for label in labels}
clus = []

while len(data) >= 1:
    matrix, updated_points, data, labels, cluster, clus = upda(data, labels, cluster, clus)
    #print("New data:", data)
    #print("Labels:", labels)
    #print("Cluster:", [cluster[label] for label in labels])
    #print("---")

print("Final Clusters:")
for i, c in enumerate(clus):
    #if len(np.unique(c)) == 100:
    print(f"Step {i+1}: {c}")
```

Tuy nhiên , khi chạy thực tế, đoạn code phân cụm không đạt hiệu quả mong đợi. Với số lượng thấp, đoạn code trả về tốt, nhưng với dữ liệu nhiều càng về cuối đoạn code không phân cụm tiếp .Mặc dù mảng cập nhật tốt sau mỗi lần lặp, cũng như đoạn mảng cập nhật về cuối đủ nhỏ, đoạn code vẫn phân cụm tốt. Điều này dẫn đến việc phân loại thất bại với số dòng lớn.

```
Step 200: [1, 3, 3, 3, 1, 3, 3, 3, 3, 10, 11, 3, 11, 3, 15, 3, 19, 3, 19, 3, 19, 22, 23, 22, 25, 3, 19, 3, 19, 30, 3, 32, 3, 3
4, 25, 3, 25, 38, 39, 40, 41, 34, 41, 44, 39, 3, 47, 48, 49, 49, 51, 52, 53, 52, 3, 56, 57, 58, 59, 60, 61, 61, 63, 64, 65, 6
6, 67, 68, 66, 68, 71, 72, 73, 74, 75, 3, 3, 75, 3, 80, 3, 82, 80, 3, 85, 86, 87, 3, 87, 90, 3, 92, 3, 92, 101, 3, 3, 3, 101,
3, 101, 102, 103, 3, 3, 101, 107, 3, 101, 102, 3, 3, 101, 3, 115, 116, 101, 3, 119, 3, 3, 101, 3, 124, 3, 3, 127, 128, 129, 13
0, 131, 130, 3, 3, 131, 136, 131, 3, 139, 3, 131, 3, 3, 3, 145, 3, 3, 3, 149, 150, 151, 152, 153, 154, 155, 156, 131, 3, 131,
3, 3, 3, 131, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 176, 3, 178, 3, 3, 181, 3, 183, 3, 181, 3, 187, 3, 189, 190, 187, 3, 3, 3,
3, 3, 3, 3, 199, 3]
```